CRESST REPORT 827

UNSUPERVISED ONTOLOGY GENERATION FROM UNSTRUCTURED TEXT

APRIL, 2013

Hamid Mousavi

Deirdre Kerr

Markus R. Iseli

National Center for Research
on Evaluation, Standards, & Student Testing

UCLA | Graduate School of Education & Information Studies

Unsupervised Ontology Generation from Unstructured Text

CRESST Report 827

Hamid Mousavi, Deirdre Kerr, and Markus R. Iseli
CRESST/University of California, Los Angeles

April 2013

National Center for Research on Evaluation,
Standards, and Student Testing (CRESST)
Center for the Study of Evaluation (CSE)
Graduate School of Education & Information Studies
University of California, Los Angeles
300 Charles E. Young Drive North
GSE&IS Bldg., Box 951522
Los Angeles, CA 90095-1522
(310) 206-1532

The work reported herein was supported by grant number OPP1003019 from The Bill and Melinda Gates Foundation with funding to the National Center for Research on Evaluation, Standards, and Student Testing (CRESST).

The findings and opinions expressed in this report are those of the authors and do not necessarily reflect the positions or policies of The Bill and Melinda Gates Foundation.

To cite from this report, please use the following as your APA reference: Mousavi, H., Kerr, D., & Iseli, M. R. (2013). *Unsupervised ontology generation from unstructured text* (CRESST Report 827). Los Angeles: University of California, National Center for Research on Evaluation, Standards, and Student Testing (CRESST).

TABLE OF CONTENTS

UNSUPERVISED ONTOLOGY GENERATION FROM UNSTRUCTURED TEXT

Hamid Mousavi, Deirdre Kerr, and Markus R. Iseli
CRESST/ University of California, Los Angeles

Abstract

Ontologies are a vital component of most knowledge acquisition systems, and recently there has been a huge demand for generating ontologies automatically since manual or supervised techniques are not scalable. In this paper, we introduce *OntoMiner*, a rule-based, iterative method to extract and populate ontologies from unstructured or free text. OntoMiner transforms text into a graph structure called a *textGraph* in which nodes are candidate terms and words from the text and edges are grammatical, semantic, and categorical relations between nodes. OntoMiner iteratively uses graph pattern rules over the textGraphs to mine ontological information and at the end of each iteration, based on the newly found information, OntoMiner improves the existing ontology. Our preliminary experiments indicate that OntoMiner achieves up to 93.4% accuracy, which to our knowledge exceeds the accuracy levels of previous work.

Introduction

By introducing *concepts* and their *relations*, *ontologies* provide a commonly understandable structure to represent information, which facilitates the processes of sharing, reusing, and analyzing domain knowledge in knowledge-based applications (Gruber, 1993). Despite the vast demands for ontologies, the reliance on manual or supervised methods often makes existing approaches impracticable and not scalable, as with Bourigault (1992), Voutilainen (1995), Pantel and Lin (2001), Drouin (2003), Snow (2006), Wu and Weld (2008), and Suchanek, Sozio, and Weikum (2009). Although some successful works such as Banko et al. (2007) and Poon and Domingos (2010) have been initiated to extract ontologies from unstructured text, their accuracy still needs improvement. As stated in Poon and Domingos (2010), none of the existing techniques can achieve a higher accuracy than 91%.

To address the need to increase accuracy, we introduce an NLP-based method, called *OntoMiner*, to automatically generate ontologies from unstructured text. OntoMiner is implemented over a recently proposed text mining framework called *SemScape* (Mousavi, Kerr, & Iseli, 2011a and 2011b) which is briefly described in the next section. In OntoMiner, using the grammatical structure of the sentences, we transform the text into a weighted hyper graph called *textGraph* which is a rich structure containing the terms and words together with their grammatical relations. We will explains the technique employed to mine terms from parse trees of sentences and generate relations between these terms using tree-based patterns (called *Tree Domain* or *TD* Rules). Note that textGraphs may also include semantic and categorical links

which are out of the scope of this report. OntoMiner also uses graph patterns (called *Graph Domain* or *GD* Rules) to mine ontological relations (e.g. *part_of* and *type_of*) between terms from the textGraphs and aggregates their weights and frequencies. Using these relations, OntoMiner detects new concepts and aliases and adds them to the current ontology. Finally, if the current ontology changes, textGraphs will be regenerated and the same approach will be repeated to find more concepts and relations.

To motivate our work and to understand why we follow the aforementioned approach, consider the following sentence:

> **Motivating Example**: "*An algebraic equation, such as linear and nonlinear equations, is an expression that contains variables and a finite number of algebraic operations.*"

As can be seen, this sentence offers several concepts such as "*algebraic equation*", "*equation*", "*linear equations*", "*nonlinear equations*", "*equations*", etc. It also offers some ontological relations such as "*linear equations*" is "*type of* " "*algebraic equation*", "*algebraic equation*" is "*type of* " "*expression*", etc. While these pieces of information are usually very hard to extract using keyword-based machine learning techniques over small textual datasets, they can be mined using NLP-based techniques with less effort.

The argument against NLP-based techniques is that they suffer from high delay as well as from not being fully automatic. The machine learning (ML) community's claim is that these approaches rely on slow text parsers mostly using manually generated rules/patterns, in which both pattern generation and pattern matching may be time consuming processes. Nevertheless, we believe that NLP-based techniques have a great potential which has not yet been exploited for the following reasons: 1) As opposed to ML-based techniques, NLP-based ones can perform on much smaller datasets, which to some extent compensate for the high delay issue. 2) They can deal with noisier datasets more easily due to their support of linguistic exceptions and ambiguity. 3) Due to their nature, NLP-based techniques can much more easily utilize distributed systems such as MapReduce (Dean & Ghemawat, 2008), since the ML-based techniques are usually recursive processes which require a lot of data exchange between nodes in the distributed environment.

In addition to these features, OntoMiner takes advantage of two types of patterns/rules: Tree Domain (TD) and Graph Domain (GD) patterns/rules to extract information from annotated parse trees and textGraphs respectively. Using TD rules, we mine more meaningful candidate terms from parse trees than most of the existing works, and using GD rules, we connect the candidate terms through ontological relations. Moreover, unlike most of the existing works, OntoMiner accepts candidate terms as new concepts based not only on their frequency and

probability, but also on their connection to existing concepts. Our experimental results indicate that even for the cases with a very small seed, OntoMiner achieves up to 93.4% accuracy which shows 2.4% improvement over one of the best existing works (Poon & Domingos, 2010).

The Text Mining Framework

Since OntoMiner is built on top of the SemScape framework (Mousavi, Kerr, & Iseli, 2011a, 2011b), we briefly introduce the framework and its features in this section. To prepare a given text (corpus), say τ, for mining purposes, SemScape uses a probabilistic parser[1] to generate a few most probable parse trees (PTs) for each sentence in τ. One such parse tree for our motivating example is shown in Figure 1 which shows each word tagged with a unique ID to differentiate between same words used in different contexts. Additionally, an easy addressing scheme has been introduced where each node address contains its parent address plus its position in the ordered list of siblings, as shown in the left column in Figure 1.

Next, SemScape annotates each node in the PTs with information referred to as *Main-Parts* (MPs). Informally, MPs carry up hidden information from the depth of PTs to the upper nodes once during preprocessing time. As will be explored later in this section, these pieces of information reduce and simplify the required higher-level mining patterns/rules, resulting in an overall faster text mining process.

[1] Charniak: www.cfilt.iitb.ac.in/anupama/charniak.php

```
[-1]    (S
[0]       (NP
[0,0]       (NP (DT an_1) (JJ algebraic_2) (NN equation_3))
[0,1]       (, ,_4)
[0,2]       (PP
[0,2,0]       (JJ such_5) (IN as_6)
[0,2,2]       (NP
[0,2,2,0]       (JJ linear_7)
[0,2,2,1]       (CC and_8)
[0,2,2,2]       (JJ nonlinear_9)
[0,2,2,3]       (NNS equations_10)))
[0,3]       (, ,_11))
[1]       (VP
[1,0]       (AUX is_12)
[1,1]       (NP
[1,1,0]       (NP (DT an_13) (NN expression_14))
[1,1,1]       (SBAR
[1,1,1,0]       (WHNP (WDT that_15))
[1,1,1,1]       (S
[1,1,1,1,0]       (VP
[1,1,1,1,0,0]       (VBZ contains_16)
[1,1,1,1,0,1]       (NP
[1,1,1,1,0,1,0]       (NP (NNS variables_17)) (CC and_18)
[1,1,1,1,0,1,1]       (NP
[1,1,1,1,0,1,1,0]       (NP (DT a_19) (JJ finite_20) (NN number_21))
[1,1,1,1,0,1,1,1]       (PP
[1,1,1,1,0,1,1,1,0]       (IN of_22)
[1,1,1,1,0,1,1,1,1]       (NP
[1,1,1,1,0,1,1,1,1,0]       (JJ algebraic_23)
[1,1,1,1,0,1,1,1,1,1]       (NNS operations_24)))))))))))
```

Figure 1. The most probable PT for our motivating example in parenthesized format.

Although four different types of MPs are considered in SemScape, the focus in this work is on noun main parts (NMPs), which contain the MP information for noun phrase nodes in the PTs (e.g. NP, NN, NNS, etc.). The annotation is done by means of a set of tree-like patterns (also called tree domain or TD rules). One such rule to extract NMPs is as follows:

Rule 1.

```
RULE mainPartRule1 ('NMP')
{
  PATTERN:  (NP *
            (? |JJ |ADJP  )
            (? |CC  )
            (JJ |ADJP  )
            (NP |NN |NNS  ) !* )
  RESULT: < [1], [3] >
  RESULT: < [1], [0]+[3] >
  RESULT: < [1], [2]+[3] >
}
```

This rule consists of two parts: PATTERN which specifies a nested pattern we need to look for in the PTs of all sentences in τ, and RESULT which indicates how the MP information

should be generated and assigned to a node (usually the root in the pattern tree). We should add that PATTERNs are nested patterns which are more expressive than regular expressions, or equivalently, finite automata (Alur & Madhusudan, 2006). This differentiates our work from most of the existing NLP-based techniques. Moreover, the tree-based format of our patterns makes them more readable and user friendly.

For instance in Rule 1, PATTERN looks for noun phrases whose last four branches are an adjective or an adjective phrase (?|JJ|ADJP), a conjunction (?|CC), another adjective or adjective phrase (JJ|ADJP), and a noun or noun phrase (NP|NN|NNS). The first two branches are optional (indicated by a question mark). From the parse tree shown in Figure 1, it is easy to see that "*linear and nonlinear equations*" in our motivating example matches this pattern.[2] If any match is found for the PATTERN, the first RESULT in Rule 1 adds the NMPs of the forth branch ("*equations*" with address [3] in the pattern tree and address [0,2,2,3] in the matching tree) of the matching tree to the NMP list of its root (the node with address [-1] in the pattern tree and address [0,2,2] in the matching tree).

Using the same type of rules, we annotate each noun phrase in the PTs with the multi-word NMPs that they may contain. For example, the above rule's last two RESULTs indicate that for the matching trees, the combination of the first and fourth branch (as well as the third and fourth) should be considered as an NMP for the root node. That is, it suggests two terms "*linear equations*" and "*nonlinear equations*" out of the phrase "*linear and nonlinear equations*" for the node located at [0,2,2].

In the current version of SemScape, we have generated more than 130 MP rules. One of the resulting annotated PTs (which are also called *main-part trees* or *MP trees*) for our motivating example is included in Figure 2. Refer to Mousavi, Kerr, and Iseli (2011a, 2011b) to see more detail on how SemScape populates the MP information across the nodes. If the framework is fed with an ontology, say *O*, each generated NMP will be tagged as a concept if it is already listed in *O*. As an example, see the node with address [0,2,2,3] ("*equations_Cequation_10*"). Since "*equations*" is already in the initial ontology (O_0) as an alias for the concept "*equation*", SemScape tags it as a known concept with *"_Cequation*."

The generated NMPs for the nodes in the MP trees are basically used as the Candidate Terms (CTs) in the OntoMiner. These MP trees will be used to find grammatical relations between words and terms using SemScape rules that are similar in format to MP rules. An example of such a rule is shown below:

Rule 2.

[2] It also matches with "*algebraic equation*", "*finite number*", and "*algebraic operations*" in the parse tree.

```
RULE subjectToVerb
{
   PATTERN: (S
                  (NP )
                  (VP ))
   RESULT (FO1='NMP', FO3='AVMP', prob=.9):
      <[0], 'subj_of', [1]>
   RESULT (FO1='NMP', FO3='PVMP', prob=.9):
      <[0], 'obj_of', [1]>
}
```

Similar to MP rules, Rule 2 indicates that for the matching trees, the NMPs of the noun phrase (NP) should be connected to the active verb main-part (AVMP) of the verb phrase (VP) to generate a subj_of relation with weight 0.9. Moreover, the NMPs of the noun phrase (NP) should be connected to the passive verb main-part (PVMP) of the verb phrase (VP) as an obj_of link. Note that, with the assistance of MP information, this single rule can catch most of the subj_of and obj_of relations without needing to know the lower level structure of the parse trees at nodes NP and VP. This is actually one of the most important gains in the SemScape framework, since having MP information decreases the number of required patterns/rules as well as simplifies them.

As you can see in Rule 2, each generated link has a weight (indicated by keyword "prob") showing SemScape's confidence in the correctness of the link. By selecting negative values for this weight, SemScape can also support exceptions in natural languages. So far we have created 270 rules to generate relations between terms and words of MP trees. After applying these rules to the MP trees of each given sentence, the generated relations will be combined to make the final set of relations. This set can be seen as a weighted hyper graph, which we refer to as textGraph. Figure 3 partially shows an expanded version of textGraph for our motivating example. To simplify the figure, we did not include the hyper links between the nodes (e.g. both nodes algebraic_2 and equation_3 are inside hyper node algebraic equation_2_3). The combination process also handles exceptions (relations with negative weight). For more details on SemScape, refer to (Mousavi, Kerr, & Iseli, 2011a).

```
[-1] S → NMP: [algebraicequation_2_3, equation_Cequation_3}
[0]     NP → NMP: {algebraic equation_2_3, equation_Cequation_3}
[0,0]       NP → NMP: {algebraic equation_2_3, equation_Cequation_3}
[0,0,0]       DT → NMP: {an_1}
[0,0,1]       JJ → NMP: {algebraic_2}
[0,0,2]       NN → NMP: {equation_Cequation_3}
[0,1]       , → NMP: {,_4}
[0,2]       PP
[0,2,0]       JJ → NMP: {such_5}
[0,2,1]       IN → NMP: {as_6}
[0,2,2]       NP → NMP: {nonlinear equations_9_10, linear equations_7_10,equations_Cequation_10}
[0,2,2,0]       JJ → NMP: {linear_Clinear_7}
[0,2,2,1]       CC → NMP: {and_8}
[0,2,2,2]       JJ → NMP: {nonlinear_Cnonlinear_9}
[0,2,2,3]       NNS → NMP: {equations_Cequation_10}
[0,3]       , → NMP: {,_11}
[1]     VP → AVMP: {is_12}
[1,0]       AUX → AVMP: {is_12}
[1,1]       NP → NMP: {expression_Cexpression_14}
[1,1,0]       NP → NMP: {expression_Cexpression_14}
[1,1,0,0]       DT → NMP: {an_13}
[1,1,0,1]       NN → NMP: {expression_Cexpression_14}
[1,1,1]       SBAR → NMP: {contains_16}
[1,1,1,0]       WHNP → NMP: {that_15}
[1,1,1,0,0]       WDT → NMP: {that_15}
[1,1,1,1]       S → AVMP: {contains_16}
[1,1,1,1,0]       VP → AVMP: {contains_16}
[1,1,1,1,0,0]       VBZ → AVMP: {contains_16}
[1,1,1,1,0,1]       NP → NMP: {variables_17, number_Cnumber_21, ...}
[1,1,1,1,0,1,0]       NP → NMP: {variables_17}
[1,1,1,1,0,1,0,0]       NNS → NMP: {variables_17}
[1,1,1,1,0,1,1]       CC → NMP: {and_18}
…
```

Figure 2. MP tree for the PT in Figure 1. For brevity, we did not include all the MP information in the graph.

In addition to the TD rules explained earlier, we have added a new type of rules, referred to as Graph Domain (GD) rules, to the SemScape framework. GD rules, which operate in the textGraph domain rather than on the MP trees, will be explained in the section labeled *The Relation Extraction Phase*.

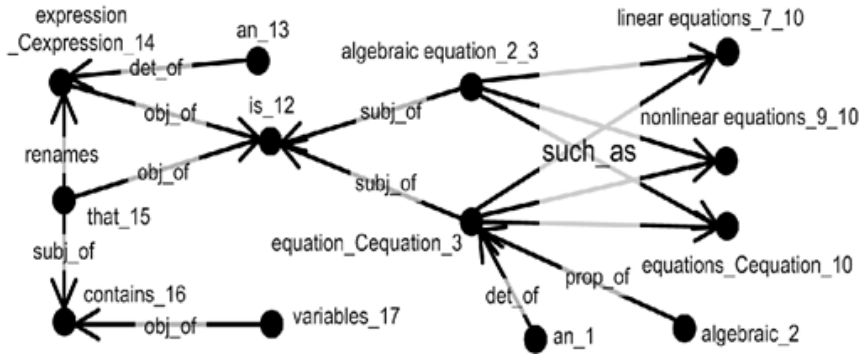

Figure 3. Part of the expanded textGraph for our motivating example. For simplifying our discussion, we do not show the weights of the relations in this graph.

Ontology Generation with OntoMiner

OntoMiner is an iterative method in which each iteration consists of two main phases: i) The relation extraction phase which generates ontological relations between the existing terms and ii) the concept extraction phase that detects new concepts and aliases using the generated relations. As shown in Algorithm 1, OntoMiner starts with an initial or seed ontology (O_0) and a corpus (τ) related to the domain of interest. Before starting the first iteration, OntoMiner initiates the SemScape framework (Line 2) and uses it to parse each sentence (Lines 5 and 6). Note that SemScape stores the parse trees to avoid reparsing the sentences.

Algorithm 1: OntoMiner**()**

```
1 : OntoMiner (τ, O0) {
2 :    SemScape = new SEMSCAPE(); /* A new instance of SEMSCAPE */
3 :    sntcs = SemScape.getSentences(τ);
4 :    O = O0
5 :    for each sntc in sntcs
6 :    PTs = SemScape.parseSentence(sntc);
7 :    while (true) /* OntoMiner's Main Iteration*/ {
8 :       rels = {};
9 :       for each sntc in sntcs{
10:          TGs = SemScape.generateTGs(PTs, O);
11:          rels += SemScape.generateOntoRelations(TGs, O);
12:       }
13:       rels = combineRelations(rels);
14:       concepts = detectOntoConcepts(rels);
15:       aliases = detectOntoAliases(rels);
16:       O = O.add(rels, concepts, aliases);
17:       if(len(concepts) == 0 && len(aliases) == 0)
18:          return O;
19:    }
20:}
```

In each iteration, OntoMiner transforms each sentence into a textGraph considering the current ontology *O* (Line 10). For each textGraph, OntoMiner generates the ontological relations between nodes (Line 11) as described in the next section. Then, it combines the generated

relations (Line 13). At this point, the relation extraction phase is finished, and a list of ontological relations between terms with their weight and frequency is created.

In the concept extraction phase, OntoMiner uses the ontological relations between concepts and non-concept terms to detect new concepts and aliases (Lines 14 and 15). The new relations, concepts, and aliases are then added to *O* (Line 16). This step (usually with looser thresholds) can also be done with human supervision, which is explained in the section labeled *The Concept Extraction Phase*. If any new concept, or an alias of an existing concept, is added to the ontology, OntoMiner will start the next iteration of mining. Otherwise, *O* is returned as the final result (Line 18).

Note that our experimental results indicate that O_0 does not need to be large at all. However, the quality of the generated ontology depends on the quality and the domain coverage of text.

The Relation Extraction Phase

To generate ontological relations between terms in the textGraph of each sentence in τ, we use graph domain (GD) patterns/rules which are a new feature in SemScape. Patterns in GD rules specify a common graph structure in textGraphs which may indicate a piece of knowledge. In our case, we are interested in using GD rules in order to find ontological knowledge, or more specifically, ontological relations. For instance, consider the link *<"algebraic equation", "type of ", "expression">*. To generate such a link from our motivating example, the graph pattern in Figure 4 can be used.

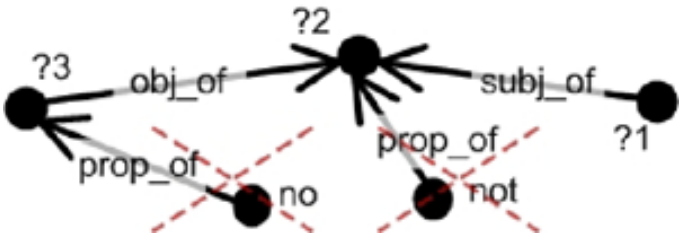

Figure 4. Pattern Graph for Rule 3.

The following GD rule specifies this pattern:

Rule 3.
SELECT (?1 "type_of" ?3)
WHERE {
?1 "subj_of" ?2.
?3 "obj_of" ?2.
NOT("not" "prop_of" ?2).
NOT("no" "prop_of" ?3).
FILTER (regex(?2, "^is^j^are^j^was^j^were^j^be", "i")) }*

The GD rules syntax is similar to SPARQL, a database query language. This makes it easier to work with, since many people are already familiar with SPARQL. Perhaps the most important difference between between our syntax and SPARQL is that GD rules operate on textGraphs in which multiple nodes with the same label are allowed, while this is not the case for SPARQL.

As shown in Figure 4, Rule 3 searches for those textGraphs in which there are two nodes (labeled ?1 and ?3), that are connected through a third node (?2) being a form of the verb "to be." The NOT part filters the negative sentences out, which is usually a challenging task in many of the existing works. Rule 3 catches the following four results from the textGraph in Figure 3: <*equation*, *type_of*, *expression*>, <*algebraic equation*, *type_of*, *expression*>, <*equation*, *type_of*, *that*>, and <*algebraic equation*, *type_of*, *that*>. The last two will later be rejected, since the term *that* is on our black list of concepts.

The main idea in OntoMiner is to create some GD rules similar to Rule 3 to generate possible ontological relations between terms in textGraphs. Note that the weights of all the relations generated from GD rules are considered to be the minimum weight among all of the edges in the matching graph. After applying the GD rules over all textGraphs, we combine the generated ontological relations from different textGraphs. The combination process is very similar to that for TD rules. Although many different techniques can be employed for combining the relations' weight, in this work we interpret the weights as our confidence of the correctness of the relations. Since we want to keep the confidence less than 1, to combine the weight of two relations with weights w_1 and w_2, OntoMiner uses $w_1+(1-w_1)w_2$ as the aggregate weight. This way each time we encounter new evidence for an existing link, we increase its confidence proportional to the new relation's weight.

After the combination phase, relations between concepts whose weight (correctness confidence) and frequency exceeds our thresholds are added into ontology *O*. The reason for having two thresholds is that relations with very high confidence but low frequency and relations with high frequency but low confidence can be rejected. The next subsection shows how OntoMiner uses the relations between concepts and non-concepts to detect new concepts.

Although in this paper, we mainly focus on the use of GD rules to extract ontological information, it is worth mentioning that GD rules can also be used to complete the textGraphs, especially for cases where several complex TD rules can be replaced with fewer GD rules.

The Concept Extraction Phase

As mentioned previously, nodes in textGraphs can be concepts, non-concept (or candidate) terms, and other words in the texts such as verbs, articles, etc. The goal in this section is to find out which of the current candidate terms are actually new concepts. Most of the previous works in this area use frequency-based techniques to accept candidate terms as concepts. This type of technique is usually error-prone and limits the accuracy of the results.

To address this issue, OntoMiner uses the ontological relations generated in the previous section to detect new concepts. The main intuition in OntoMiner is that if a candidate term is a concept in the domain of τ, it should be connected to some other concepts of the domain through one or more ontological relations. Thus, OntoMiner accepts a term as a new concept if it has a "strong" ontological relation to other existing concepts. Here, by a strong relation, we mean that the frequency and aggregate weight of the relations from the new term to any existing concepts should be greater than given thresholds. In other words, to accept candidate term CTi, OntoMiner finds all the relations having CTi at one end and a concept at the other end. Then, it combines their weight (confidence) and frequency exactly as in the previous section. At this point, high-frequent and high-confident terms will be accepted as new concepts; however, before accepting a term as a new concept, we adjust its weight and frequency using heuristics such as:

- If the term contains a stopping word (e.g. that, which, who, etc.), is a word from the black list (e.g. same, the, a, etc.), or is an attributive adjective (e.g. short, similar, etc.), we ignore it.
- If the term contains a word from our black list, an attributive adjective, a numeric (3, six, secondary, etc.), a symbol (+, , Ω, etc.), or repeated words, we will decrease its weight and frequency[3].

After these adjustments, we accept those candidate terms whose weight and frequency are larger than our pre-specified thresholds. In the unsupervised version of OntoMiner, the new concepts will directly be added to current ontology *O*. However, for the supervised version, the new concepts will be verified by a human scorer. Since precision is not much of an issue in this case, we usually use looser thresholds in the supervised version to offer more concepts and improve the recall value.

[3] Currently, they are decreased by 1% and 30% respectively.

Detecting New Aliases

A single concept may have several "names." Research shows that people use different names for the same concept more than 80% of the time (Furnas et al., 1987). Abbreviations, variations of the term (e.g. plural form), acronyms, aliases, etc. may be used to address the same concept. An important task in generating ontologies is to find these different names or *aliases* for the same concept. In OntoMiner, we use the following three techniques to do so:

- We directly use either TD or GD rules to find aliases explicitly mentioned in the text (e.g. "*a node may be called a vertex*."). This technique considers aliases as ontological relations as in the previous section.
- For many link types, say *l*, such as *type_of*, if two concepts are "strongly" connected only through *l* in both directions, they will be considered aliases (e.g. "*edge is type of side*" and "*side is type of edge*" means that side and edge are aliases.)
- If the stem of a concept is also a concept, it will be considered an alias (e.g. *sides* and *side*).

The same adjustments explained in the previous subsection will be applied to pick the final aliases. Note that we group all the aliases of the same concept and (randomly) consider one of them as the head of the group. In the next iteration of OntoMiner, at the MP annotation time, we use the head of each group for tagging any of the aliases in that group. As an example, in Figure 2 both *equation_3* and *equations_10* are tagged with the same alias (*equation*).

Implementation and Optimization

This subsection is intended to provide more details on OntoMiner's implementation and optimization.

As already mentioned, OntoMiner is implemented on top of the SemScape framework, and it can be seen as a new component of SemScape. For numerous text mining applications, this is an invaluable feature since it can automatically find the concepts and tag the nodes in the textGraphs with ontological information. This basically means that OntoMiner enables the SemScape framework to automatically adapt to new domains. As a result, this enriches the SemScape framework, and facilitates the mining task even more.

One of the simplest and perhaps most effective optimizations in OntoMiner is to avoid redoing tasks by storing and reusing intermediate results. Remember that at the end of each iteration, OntoMiner feeds the SemScape framework with the newly found concepts. Parse trees

extracted from the sentences in the first iteration are stored and reused in the following iterations. MP trees are only recalculated if they have changed due to addition of a new concept or alias, which results in OntoMiner working only on sentences that contain one or more of the new concepts.

Another important optimization that OntoMiner can utilize is to distribute its task over different processing units. According to our experiments the most time consuming part of OntoMiner is transferring text to textGraphs. Fortunately, this task is separately done for each of the sentences in τ. This essentially means that by assigning the sentences to different processing units, we can distribute the most time consuming task in OntoMiner (and perhaps many other text mining algorithms) with minimum effort. This task is best suited for the MapReduce architecture (Dean & Ghemawat, 2008) in which mappers perform relation extraction from each text graph and the reducer(s) combines them and detects the new concepts. The current implementation already supports multi-core machines, and at the moment of writing this paper we are extending it to a multi-processor environment.

The final point we need to clarify is the difference between TD and GD rules. One may observe that whatever GD rules can extract from textGraphs, TD rules can extract from MP trees. Pattern matching with TD rules is faster than that pattern matching with GD rules. However, GD rules are more intuitive for the users to write, and to extract a piece of information users need to write fewer GD rules than TD rules, which can make up for the pattern matching speed advantage with TD rules.

Experimental Results

To evaluate OntoMiner, we use a mathematics ontology manually created by CRESST with 877 concepts including their definitions. Considering these definitions, we have created 10 and 14 GD rules to generate, respectively, *type_of* (*IsA*) and *part_of* (*HasA*) relations. We have also gathered 605 definitions from Wikipedia by selecting the first 3 or 4 sentences of those pages having any of our 877 concepts as part of their title. The resulting WIKI dataset has approximately 2200 definition sentences whereas the CRESST dataset has approximately 900 sentences. Note that, to be fair in our results, we have not used any sentences from the WIKI dataset to generate the GD rules. In contrast to works such as Gregorowicz and Kramer (2006), we do not use the structured part of Wikipedia to extract any ontological information. The experiments are run on a 32bit, Intel Core 2 Due 2.53GHz CPU machine running Linux with 2GB of RAM and 4MB of cache.

Table 1

The results of generated concepts, “-spr” indicates supervised mode

DataSet	Seed size	Total #	Wrong #	Precision	Relative recall
CRESST	1	423	42	90.1%	88.4%
CRESST	877	293	23	92.1%	94.8%
CRESST-spr	1	329	0	100%	-
WIKI	1	458	30	93.4%	82.1%
WIKI	877	313	41	86.9%	86.1%
WIKI-spr	1	447	0	100%	-

Tables 1 and 2 depict the results of running OntoMiner over our two datasets, CRESST and WIKI, with two initial (seed) ontologies; one with all 877 concepts and one with only the concept "*function*." In unsupervised experiments, we fixed the weight and frequency thresholds on .98 and 4 respectively. We manually evaluated the resulting concepts, aliases, and *part_of* and *type_of* relations (columns 4 and 5 in both tables). We have eliminated the results for aliases and *part_of* relations due to the similarity of the results. As can be seen, the precision results (column 5 of the tables) are comparable with the ones in Poon and Domingos (2010). We have over 2.4% improvement with respect to them even though we are using a smaller dataset.

Table 2

The results of generated type_of relations

DataSet	Seed Size	Total #	Wrong #	Precision
CRESST	1	307	40	87.0%
CRESST	877	478	40	91.6%
CRESST-spr	1	224	13	94.2%
WIKI	1	273	34	87.5%
WIKI	877	408	24	94.1%
WIKI-spr	1	285	8	97.2%

To estimate the recall of the generated concepts, we ran OntoMiner in a supervised mode for both of our datasets (CRESST-spr and WIKI-spr in Tables 1 and 2), in which at the end of each iteration the suggested concepts and aliases are verified by a human scorer, and only the

correct ones are accepted to the current ontology. This is why the generated concept accuracy for these two cases is 100%. The weight and frequency thresholds for the supervised version are .96 and 3. Considering CRESST-spr and WIKI-spr as baselines, we estimated the relative recall values for both datasets in the sixth column of Table 1. Not surprisingly, the recall value for the CRESST dataset is higher than that for the WIKI dataset because we generated the GD rules specifically for the CRESST dataset. Moreover, one can theoretically increase both recall and precision of the CRESST dataset to 100% by adding enough rules; however, this in practice may require adding several specialized rules.

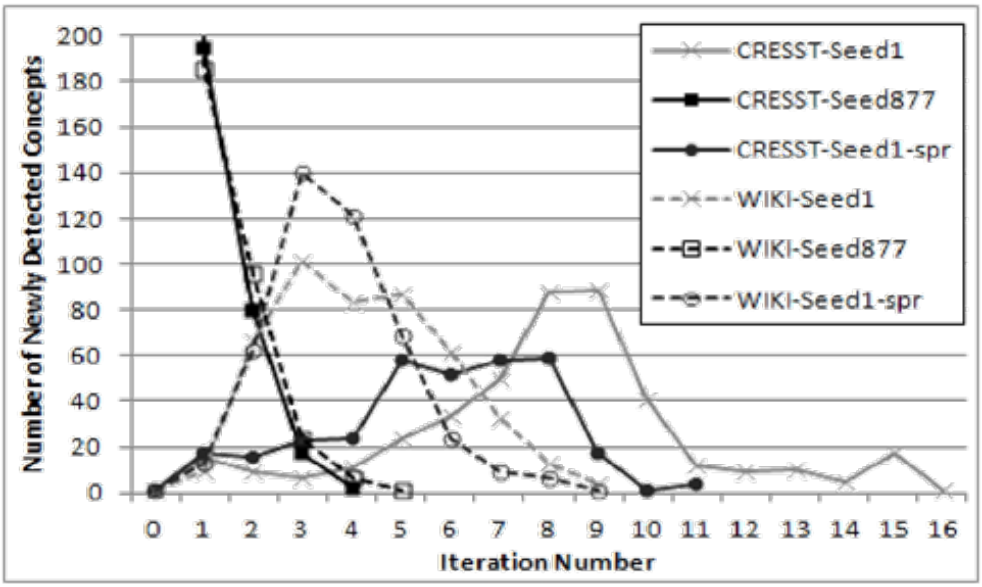

Figure 5. The number of detected concepts at each iteration for each experiment.

Note that decreasing the seed size to one concept does not dramatically lessen the accuracy of the results. Moreover, applying the same GD rules to a new text (WIKI dataset) improved the accuracy over the CRESST dataset, proving that the rules are not specialized to the original dataset. However, as shown in Figure 5, the number of iterations generally increases for smaller seeds, even though increasing the seed size does not always decrease the number of iterations.

It could be shown that by adding a few new GD rules and completing our black, stop, and attributive adjective lists, the accuracy improved even more. This claim can be verified by the high accuracy results of our supervised runs for *type_of* relations. Figure 6 shows the graph structure of *type_of* relations generated for the WIKI dataset with seed of size one. Note that in this graph if *X* is type of *Y* and *Y* is type of *Z*, the link <*X*, *type_of*, *Z*> will not show; our ontology graph is not necessarily a connected graph even considering the *part_of* links.

It is worth mentioning that all of our experiments are run in less than seven hours which is still acceptable for a one-time preprocessing task. However, this time performance can be easily improved using a distributed environment as discussed in the section labeled *Implementation and Optimization*. Indeed, using two processing cores instead of one core in our experiments has almost doubled the speed, showing OntoMiner's great parallelization capability.

Related Work

Machine learning (ML) techniques have been used to generate ontologies for some time. Loh, Wives, and Oliveira (2000) used fuzzy reasoning to calculate the likelihood of a term to be a concept. Pantel and Lin (2001) proposed a language independent technique in which they extract high frequency two-word terms, extending them subsequently to multi-word terms using statistical techniques. In Quan, Hui, Fong, and Cao (2004) and Tho, Hui, Fong, and Cao (2006), Quan et al. incorporated fuzzy logic into Formal Concept Analysis or FCA (Ganter & Willi, 1999) to automatically extract ontologies. Another approach, based on fuzzy logic, was proposed by Lee, Kao, Kuo, and Wang (2007). Parameswaran, Garcia-Molina, and Rajaraman (2010) used some variations of association rule mining techniques to find frequent terms and words from logs and tags.

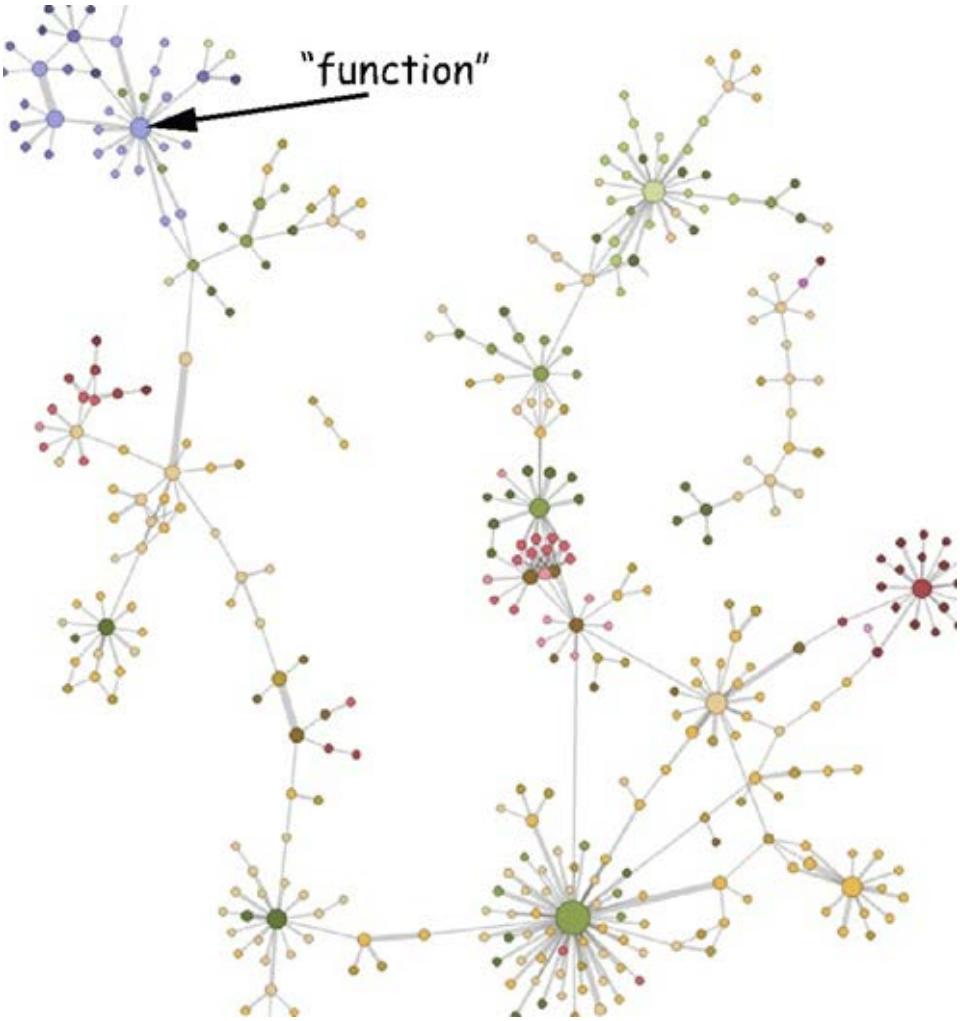

Figure 6. Part of the generated ontology for CRESST dataset and seed of size one (*type_of* relations only). Nodes with the same color are generated in the same iteration.

Most of the above techniques suffer from at least one of the following issues: They either need large corpora in order to provide highly accurate results, or they cannot handle reordering of the words to make new terms (e.g. "*linear equations*" from "*linear and non-linear equations*") mostly due to ignoring the grammatical structure of the corpus, or they rely on structured or semi-structured datasets which limits their coverage.

Other well studied approaches contain supervised techniques to extract ontological information. Bourigault (1992) used a two-phase algorithm to first analyze the given (French) corpora for candidate terms and then parse them to find the final terminological units using their grammatical structures and the position of words in the maximal-length term. Based on this work, Drouin proposed a hybrid technique to extract terms from POS-tagged texts and then filter them based on some statistical techniques (Drouin, 2003). Similar techniques are proposed by

Voutilainen (1995) and Maedche and Staab (2000). Unfortunately, most of these approaches are not scalable, since a human validation is required to verify the final results. Moreover, a limited and fixed number of regular expressions (REs) are usually used to find the terms, which limits their precision and recall.

In recent years, more unsupervised text mining methods are appearing. Lin and Pantel (2001) automatically found similar paths in dependency trees. These similar paths form a (binary) relation structure between terms and can be used in ontology generation systems. Banko et al. (2007) introduced a new extraction paradigm, called OIE, in which some relations are extracted from the corpus in preprocessing time to facilitate the main process of text mining. Later, Poon and Domingos (2010) proposed OntoUPS which is more robust to noise with respect to OIE. OntoUPS is based on a semantic parser called UPS (Poon & Domingos, 2009). UPS converts dependency trees to quasi-logical forms, and uses recursive reductions to abstract out syntactic variation. Krishnamurthy and Mitchell (2011) used the relations extracted from NELL's knowledge-base (Carlson et al., 2010) to extract concepts and aliases. We believe that the precision and recall of these new techniques still need improvement. This is mainly because they do not fully utilize the grammatical structure of the text and do not explicitly support linguistic exceptions, linguistic ambiguity, and negative sentences. Moreover, their clustering-based technique requires them to use large corpora in order to achieve higher precision.

Conclusion

In this paper, we introduced the OntoMiner system which automatically generates ontologies from unstructured free text using a fixed set of rules. OntoMiner is a rule-based system which takes advantage of two types of rules: tree-domain (TD) and graph-domain (GD) rules. TD rules are used to convert text to a graphical structure called textGraph and GD rules are used to mine ontological relations among nodes in the textGraph. These relations are later used to detect new concepts and iteratively complete the ontology. Although OntoMiner already outperforms one of the best existing automatic ontology generators, it still shows a lot of potential for improvement. Currently, in addition to completing our GD rules and finding more general ontological link types to improve the accuracy of the system, we are working on distributing OntoMiner's task to several machines to improve its time performance and scale it up for larger text amounts. We have also been working on pronoun and co-reference resolution techniques to improve the textGraphs quality.

References

Alur, R. & Madhusudan, P. (2006). Adding nesting structure to words. Developments in Language Theory. In (Vol. 4036, pp. 1–13): Springer Berlin/Heidelberg.

Banko, M., Cafarella, M. J., Soderl, S., Broadhead, M., Etzioni, O. (2007). Open information extraction from the Web, Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), (pp. 26070-2676), Hyderabad, India.

Bourigault, D. (1992). Surface grammatical analysis for the extraction of terminological noun phrases. In Proceedings of the Fifteenth International Conference on Computational Linguistics, pages 977-981.

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Jr., E. H., & Mitchell, T. (2010). Toward an architecture for never-ending language learning. In Proceedings of the Conference on Artificial Intelligence (AAAI), (pp. 1306–1313), AAAI Press.

Dean, J. & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107–113.

Drouin, P. (2003). Term extraction using non-technical corpora as a point of leverage. TERMINOLOGY, 9:99–116.

Furnas, G. W., Landauer, T. K., Gomez, L. M., & Dumais, S. T. (1987). The vocabulary problem in human-system communication. Commun. ACM, 30(11):964–971.

Ganter, B., & Wille, R. (1999). Formal concept analysis - mathematical foundations. Springer.

Gregorowicz, A., & Kramer, M. A. (2006). Mining a large-scale term-concept network from wikipedia. Technical report, Mitre.

Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, (6, pp. 199–220).

Krishnamurthy, J., & Mitchell, T. M. (2011). Which noun phrases denote which concepts? In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1, HLT '11, pages 570–580, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lee, C.-S., Kao, Y.-F., Kuo, Y.-H., & Wang, M.-H. (2007). Automated ontology construction for unstructured text documents. Data & Knowledge Engineering., 60(3):547–566.

Lin, D. & Pantel, P. (2001). Dirt @sbt@discovery of inference rules from text. In KDD, pages 323–328.

Loh, S., Wives, L. K., & De Oliveira, J. P. M. (2000). Concept-based knowledge discovery in texts extracted from the web. SIGKDD Explor. Newsl., 2(1):29–39.

Maedche, A. & Staab, S. (2000). Semi-automatic engineering of ontologies from text. In Proc. of 12th Int. Conf. on Software and Knowledge Engineering, Chicago, IL.

Mousavi, H., Kerr, D., & Iseli, M. (2011a). A new framework for textual information mining over parse trees. In ICSC, (pp. 185–188).

Mousavi, H., Kerr, D., & Iseli, M. (2011b). A new framework for textual information mining over parse trees. In (CRESST Report 775). University of California, Los Angeles.

Pantel, P. & Lin, D. (2001). A statistical corpus-based term extractor. In Canadian Conference on AI, pages 36–46.

Parameswaran, A. G., Garcia-Molina, H., & Rajaraman, A. (2010). Towards the web of concepts: Extracting concepts from large datasets. PVLDB, 3(1):566–577.

Poon, H. & Domingos, P. (2009). Unsupervised semantic parsing. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1, EMNLP '09, pages 1–10, Stroudsburg, PA, USA. Association for Computational Linguistics.

Poon, H. & Domingos, P. (2010). Unsupervised ontology induction from text. In ACL, pages 296–305, 2010.

Quan, T., Hui, S., Fong, A., & Cao, T. (2004). Automatic generation of ontology for scholarly semantic web. Proceedings of The Semantic Web–ISWC, (pp. 726–740).

Snow, R. (2006). Semantic taxonomy induction from heterogenous evidence. In In Proceedings of COLING/ACL 2006, (pp. 801–808).

Suchanek, F. M., Sozio, M., & Weikum, G. (2009). Sofie: a self-organizing framework for information extraction. In WWW, (pp. 631–640).

Tho, Q. T., Hui, S. C., Fong, A. C. M., & Cao, T. H. (2006). Automatic fuzzy ontology generation for semantic web. IEEE Trans. on Knowl. and Data Eng., 18(6):842–856.

Voutilainen, A. (1995). Nptool, a detector of english noun phrases. CoRR, cmp-lg/9502010.

Wu, F. & Weld, D. S. (2008). Automatically refining the wikipedia infobox ontology. In WWW, (pp. 635–644).